Search

Agents

Agents interest of Complete Process (simple system)

Agents interest of what will happen after ending the Process of find best solution. (complex system)

[1] Reflex agent (simple system)

a) choose action based on Current Percept.

b) may have memory or model of world's Current state.

c) Do not Consider future Consequence of their actions.
"Do not ask what if"

d) Consider how the world is (Cannot care about future actions)

e) Can it be rational (عاقلة) ?
↳ yes, if it is Provided by if conditions.

[2] Planning Agents (complex system)

a) Ask "what-if"

b) Decisions based on (hypothesized) consequences of actions.

c) must have model of how world evolves in response to actions.

d) must formulate a goal (test) ⇒ must reach the goal.

e) consider how the world would be.

Planning	Replanning
→ Agent make plan according to search. → It take period of time to plan then take decision according to plan. → Plan only one time	→ Agents make plan for the first step according to search then take decision to complete first step. then → It replan for next step according to new search and take new best decision to start work for next step. → plan for each step ↳ no. of plans = no. of steps.

*optimal vs Complete Planning

↳ Agents not only reach goal or take action but also it want to get optimal solution of problem. (search for best way to goal or minimum time to reach goal)

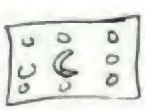
* Search Problems :

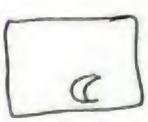
↳ Process to take right decision.

It Consists of

1) state space (world) : All Possible cases of the model.

ex mouse want to eat g piece of cheece.

 → mouse eat Piece on center

 → eat the last piece.

2) A Successor function (actions, Costs)
↳ For any state what action can I take and what cost of taking it.

3) A start state an goal test

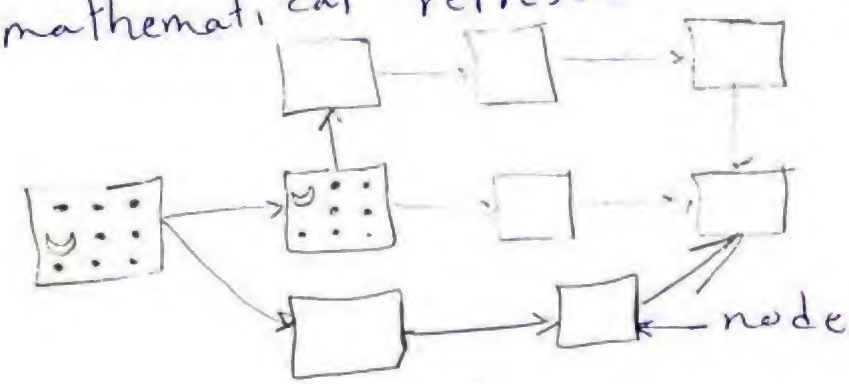
start state: state from which i can start working.

goal test: the final state.

→ Solution of search problem is sequence of action which transforms start state to goal state.

* Uniformed search methods

① state space ~~representation~~ graph of search problem.
↳ mathematical representation

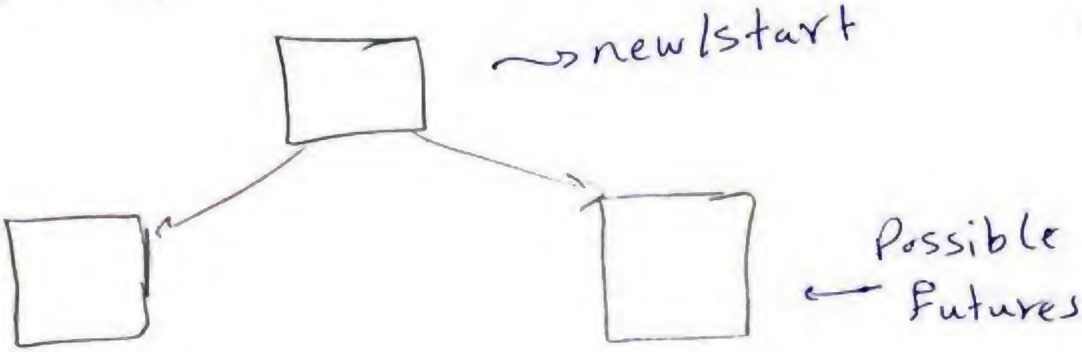


- Nodes → world configuration.
- Arc → successors (action results)
- goal test is set of goal nodes.

→ here we represent needed states only.
→ we can rarely build this full graph.
→ there is no start state. but goal test

② search trees

↳ part of search graph.



Search tree

→ start state is the root node.

→ children correspond to successors.

→ we can never actually build whole tree.

× searching with search tree

a) try to expand as few tree nodes as possible.

b) maintain a fringe of partial plans under consideration.

c) try to expand as few tree nodes as possible

(General Tree search)

Frings : all of plans that may yet work.

→ all ways that can reach me from start to goal during world state search tree.

→ one of this frings is my solution (optimal solution)

Expansion

↳ Picking some thing out of the fringe and if it is not a goal already.

→ Process of expanding new state from current state on one of frings. we can't do this process if we reach goal.

---

* Exploration strategy

↳ what fringe nodes do you explore next?

↳ what will happen next?

* Main Question:-

↳ which fringe nodes to explore to reach goal?

Depth-first search

strategy: expand deepest node first

Implementation: fringe is LIFO stack.

solution : left most solution.

Search Algorithm Properties

→ For any Algorithm we check for 4 Properties:-

1. Complete:
   ↳ Ability to reach goal if it is exist.

2. optimal:
   ↳ Ability to reach to best solution.

3. Time Complexity
   ↳ time to reach to solution
   ↳ time to expand all nodes of one fringe equal to time to expand one node of fringe * no of fringe nodes.

4. space Complexity
   ↳ size of fringe in memory. size of nodes of fringe in stack.

* no. of nodes in entire treee.

$$1 + b + b^2 + \dots + b^m = O(b^m)$$

→ branching factor:- start node can make a branch for b number of children.

Depth first Properties

[1] Time Complexity

→ what nodes DFS expand?
- → some left prefix of tree.
- → Could Process the whole tree!
  - → worst case expand all nodes (right most)
  - → best case expand no nodes (start $\equiv$ goal)

worst case = $O(b^m)$, best case = $O(1)$

[2] space Complexity

- → only has siblings on path to root, so $O(b^m)$
- → worst case we get solution on last tier.

[3] Complete

→ m could be finite, if we prevent cycles.

→ solution can only be found if

1. it is exist
2. finite Algorithm.

[4] optimal

→ no, it finds left most solution regardless of depth or cost.

[2] Breadth-First (BFS)

Strategy: expand a shallowest node first.

Implementation: fringe is FIFO queue.

Solution: shallowest solution (shortest fringe)

* DFS & BFS

→ BFS will outperform DFS when:

1) need less time complexity.
2) need shallowest solution.
3) need optimal solutions for costs equal to 1.
4) need complete solutions.

→ DFS will outperform BFS when:

1) most solutions at left side of tree.
2) all solutions at the last level.
3) need less space complexity.

[3] uniform Cost

strategy: expand a cheapest node first.

Implementation: fringe is priority queue

solution: cheapest solution.

→ expand node of cheapest code.

*** Advantages**

→ complete and optimal.

*** Disadvantages**

1. no information about goal location.

← ممكن تكون وصلت لل (goal) بس مش عارف تعمله (expand)
لأنه في (node) تانية أرخص.

2. explores options in every direction.

↳ no determined direction on our work.

* The one Queue: Priority queues

→ Conceptually, all frings are Priority queues.

→ For DFS and BFS you can avoid the log(n) overhead from actual Priority queue with stacks and queues.

→ Can even Code one implementation that takes variable queuing object.

Informed search

↳ we need to Know information about goal location to Know if I work on correct way or not.

→ we have one Function, 2 search Algorithms:

1) Heuristics

↳ function takes state of state space, and give number which represent how far the goal location from my location to doing process.

* Greedy search

↳ Search Algorithm use idea of Heuristic.

↳ It is not an optimal search Algorithm

* A* ~~Algorithm~~ search

↳ It collects all last search Algorithms ideas to get very good search Algorithm.

* ~~Graph search~~

Recap search

1) Search Problem

a) states (configuration of world) b) actions & costs.

c) successor function (say how states respond to actions)

d) start state & goal state.

2) search tree

for reaching states.

a) nodes : Plans

b) Plans have costs (sum of actiin costs)

3) search Algorithm

a) systematically builds search tree.

b) chooses an ordring of the fringe.

c) optimal : find the least-cost plans.

EX: Pancake Problem

Problem: need to arrange Pancake from big to small.

states: Shape of Pancakes during flipping to reach goal.

Costs: no. of Pancakes flipped.

start state,  goal state: 

Algorithm: we can use UCS algorithm according to cost also we can use DFS or BFS.

Search Heuristics

heuristic is

a) function that estimates how close a state to goal.

b) Designed for particular search Problem.

c) we make it every step to see if i close to goal or not.

d) every search Problem need different heuristic According to natural of Problem.

ـ اكمال المثال:

Heuristic: no. of largest Pancake that is still out of place.

Greedy search

strategy: expand the nodes that seems closest to goal according to heuristic.

Heuristic: estimate the distance to nearest goal for each state.

Implementation: fring is priority queue ξ:

solution: lest heuristic solution.

→ It doesn't care about cost.

→ cares only about heuristic

~~[illegible]~~

common case: Best-first takes you to wrong goal.

worst case: like badly-guided DFS.

solution → Combine UCS and Gready search.

A^* search

→ combining UCS and Gready Search.

* uniform cost orders by path cost or backward cost $g(n)$.

* Gready search " by goal proximity or forward cost $h(n)$.

↳ should we stop when we enqueue goal?

↳ No only stop when we dequeue goal.

Admissibility

* Inadmissible (Pessimistic) heuristics break optimality by trapping good plans on fringe.
* Admissible (optimistic) heuristics slow down bad plans but plans never outweigh true costs.

Admissible heuristic

→ A heuristic h is admissible if:

$$0 \leq h(n) \leq h^*(n) \quad \rightarrow \quad h^*(n) = g(n)$$

→ $h^*(n)$ is true cost to nearest goal

← مش هتقدر نوصل لل (optimal solution) لو $h(n) > h^*(n)$

* A^* applications

* video games.
* language analysis.
* Machine translation
* speech recognition.
* Robot motion Planning.
* resource Planning Problems.

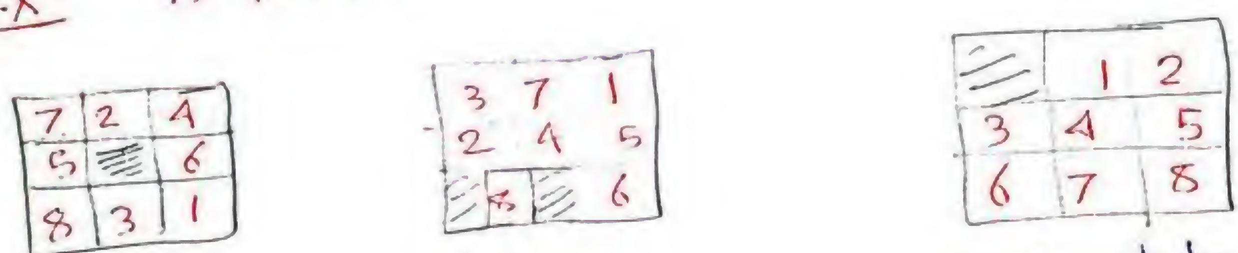
* Creating Admissible heuristics

→ most of work in solving hard search Problems optimally is in Coming up with admissible heuristics.

→ often, admissible heuristics are solutions to relaxed problems where new actions are available.

→ inadmissible heuristics are often useful too.

Ex 8 Puzzle



start state Action Goal state

states? all cases that i can move any number from its location to another one if it is next to free space

How many states → 8! =

actions? moving number to free space next to it.

* no. of successors from start state?
maximum 4 moves , minimum 2 moves
* Cost should be? 1 every time
* Heuristic : no. of tiles misplaced.
$h(start) = 8$
* why it is admissible?
1) There is relaxed - problem heuristic
(easiest solution but need more work)

2) Direct move
↳ move every number from start state to final state.
h = no. of moves.
$h(1) = 3$ (moves to reach to its correct state at goal state)
$h(2) = 1$, $h(3) = 2$ an so on.
→ How Total $h = 18$ from start to goal.
Admissible if $0 \leq h(n) \leq g(n)$

⇝ How about using actual cost as heuristic?

$f(n) = h(n) + g(n) = 2g(n) = 2h(n)$

if $g(n) = h(n)$

↳ we go back to UCS ~~to~~ by double cost value trade off: complex work, neglect heuristic

with A^* → trade off between quality of estimate and work per-node.

* Trivial heuristic

- bottom of lattice is zero heuristic
- top " " is the exact "

if $(h = 0)$ no heuristic ⇒ go back to (uniform cost search)

Dominance: $h_a \leq h_c$

$\forall n : h_a(n) \geq h_c(n)$ for all nodes

we choose $h_a(n)$ (best one)

⇒ Heuristics from a semi-lattice

↳ max of admissible heuristics is admissible

$h(n) = \max(h_a(n), h_b(n))$

~~we choose~~

* Graph search

↳ failure to detect repeated states can cause exponential more work.

idea: never expand state twice.

Implementation:

* tree search + set of expanded states (closest set)
* Expand search tree node by node, but never expand node twice.
* Before expanding a node, check if it is expanded before or not.
* if not now, skip it, if new add to closest set.

Important

* store closest set as a set, not a list.
* Complete ⇝ if goal exists, we will find it.
* optimal ⇝ no.

	Depth first search	Breadth first search	Uniform Cost search
nodes expanded	→ Some left prefix of the tree. → Could process the whole tree. → if m is finite, take time $O(b^m)$	→ Processes all nodes above shallowest solution. → let depth of shallowest solution be s. → Search takes time $O(b^s)$	→ Process all nodes with cost less than cheapest solution. → takes time $O(b^{C^*/\varepsilon})$ C^* → sol. costs
space that fringe take	$O(b^{m})$	$O(b^s)$	$O(b^{C^*/\varepsilon})$
Complete	only if we prevent cycles	s → finite Yes	Assume best sol. has finite cost and min. ε ↳ Yes
optimal	No	only if costs are all 1	Yes

→ heuristic

* Function estimates how close a state is to a goal.
* Designed for particular search problem.
* Ex: Manhattan distance.

Greedy search

* **strategy** expand a node that you think is closest to a goal state.

↳ Heuristic ⇒ estimate of distance to nearest goal for each state.

* **Common case**

↳ Best-first takes you to wrong goal.

A* search

↳ only stop when we dequeue a goal.

→ Admissible Heuristics

→ heuristic h is admissible (optimistic) if

$$0 \leq h(n) \leq h^*(n)$$

where ⇒ $h^*(n)$ → is true cost to nearest goal.

UCS vs A^* Contours

UCS → expands equally in all "directions".

A^* → expands mainly toward the goal but does hedge its bets to ensure optimality

A^* Applications

video games, language analysis, speech recognition, Machine translation, Pathing.

→ search Problem consists of

1) state space
2) successor function (with actions, costs)
3) start state and goal test.

solution → is sequence of actions (a plan) which transform start state to goal state.